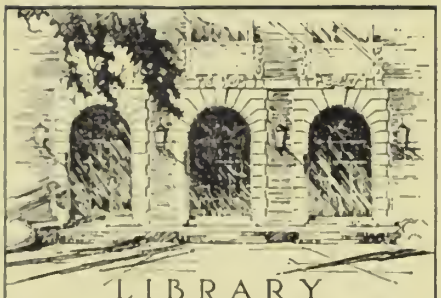


Report No. 305

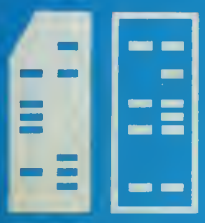
Math

An Implicit Enumeration Program for Zero-One Integer Programming

by

T. Ibaraki
T. K. Liu
C. R. Baugh
S. Muroga

January 6, 1969



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. 305

An Implicit Enumeration Program for Zero-One Integer Programming

by

T. Ibaraki
T. K. Liu
C. R. Baugh
S. Muroga

January 6, 1969

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

ABSTRACT

This paper describes a theoretical background and computational experience of a recently implemented computer program for solution of a zero-one integer programming which is called ILLIP (ILLinois Integer Programming.) The program is based on the implicit enumeration method.

In order to speed up the original implicit enumeration method of Balas,[1] Glover,[10] Geoffrion,[8] and Fleischmann[6] a few gimmicks are proposed. Among those the biggest speed-up is attained by introducing a new column which works as a tag for each inequality indicating whether it should be checked or not for the current partial solution generated during the implicit enumeration procedure. In addition to the well known conditions in which a variable can be underlined, a new condition was found. A new concept of pseudo-underlining is also proposed.

ILLIP was applied to many problems to test computational efficiency. Logical design of optimum digital network, for which our program was initially designed, was very efficiently solved. Other problems which have been known in literature also were solved. The method worked better than those proposed by different authors for some problems but not for other problems.

I. INTRODUCTION

The implicit enumeration algorithm for zero-one integer programs has been formulated by various contributors which includes Balas[1], Glover[10], Geoffrion[8], Fleischmann[6] and Lemke-Spielberg[15] among others. This wide-spread attention is mainly due to the fact that the implicit enumeration method appears to be quite efficient for a number of problems. In addition, a "reasonably good" feasible solution is often available if the algorithm is halted prior to normal termination, and addition and subtraction are the only arithematic operations required, thus eleminating round-off problems due to multiplication and division.

Our computer program which is called ILLIP (ILLinois Integer Programming Code) was initially designed for a particular class of problems encountered in the logical design of optimal digital network.[19] [20] [3]. These problems may be characterized by:

(a) more inequalities than variables and thus creating a relatively "small" solution region;

(b) A very sparse coefficient matrix with non-zero coefficients most of which are 1 and -1, and;

(c) an objective function with coefficients, 0 and 1.

Considering these properties, we adopted and modified the implicit enumeration algorithm. Our main effort was to speed up the enumeration procedure which is roughly based on the backtracking scheme due to Glover[10] and the checking procedure of inequalities publicized by Fleischmann[6]. Several new techniques were incorporated for the speed-up and they will be explained in detail. In addition, a new concept called "pseudo underline scheme" was introduced.

One of the typical features of any integer programming algorithm, however, is its erratic behavior for a given problem. An algorithm which can solve a problem very efficiently may not be very efficient for other problems.

Although our approach is initially designed for such a problem as mentioned above, it also seems to be reasonably efficient for other problems taken from literature as will be shown in Section IV. However, a further speed-up could be usually obtained by making use of intrinsic properties of a given problem. Some gimmicks are tested for the traveling salesman problems and the covering problems. The result for these gimmicks also will be given in Section IV.

One of the largest problems successfully solved had 258 variables and 674 inequalities. The first feasible solution was obtained in less than 20 seconds, an optimum feasible solution in less than 30 seconds, and total computation time of less than 15 minutes on the IBM 360 model 75I. The largest problem was with 395 variables and 1012 inequalities, and took 5 minutes and 15 seconds on the same machine. For this problem, the algorithm was modified to make use of the structure of the given problem.

II BASIC IMPLICIT ENUMERATION

The integer linear programming problem is generally stated as:

$$\begin{aligned} &\text{minimize} && \vec{c} \cdot \vec{x} \\ &\text{subject to} && \vec{b} + A\,\vec{x} \geq 0 \end{aligned} \tag{2.1}$$

where $\vec{c}$ is a vector of n non-negative coefficients, $\vec{b}$ is a column vector of m constants, A is a $m \times n$ matrix and $\vec{x}$ is a column vector of the n zero-one variables.

If the original integer linear programming problem is not given in the above form, we can easily transform the problem into the above form.

The implicit enumeration algorithm for integer programming is an enumeration scheme which exhaust all possible variable assignments without examining all of the 2^n combinations of an n variable problem. It can be interpreted as a special type of " branch and bound method"[14]. An excellent explanation of the basic implicit enumeration scheme is given by Geoffrion[8].

Before we give the details of our algorithm we need some definitions and notations. Any $\vec{x}$, whose coordinates are 0 or 1, is called a solution. A solution that satisfies the constraints

$$\vec{b} + A\,\vec{x} \geq 0$$

is called a feasible solution, and a feasible solution that minimizes $\vec{c}\,\vec{x}$ is called an optimal (feasible) solution.

A partial solution S is defined as an assignment of the values one and zero to a subset of n variables. Any variable not assigned a value by S is called free. We adopt the notational convention that the symbol j denotes $x_j = 1$ and the symbol -j denotes $x_j = 0$. Hence, if n = 5 and

S = {3,5, -2}, then x_3=1, x_5=1 x_2=0 and x_1 and x_4 are free. It will be seen that the order in which the elements of S are written will be used to represent the order in which the variables corresponding to the elements are set to 1 or 0 by the algorithm. A completion of a partial solution S is defined as a solution that is derived from S by specifing all free variables.

Implicit enumeration involves generating a sequence of partial solutions and simultaneously considering completions of each partial solution. As the calculation proceeds, feasible solutions are discovered and the best (minimum $\vec{c}\cdot\vec{x}$) one yet found is stored as an incumbent.

The entire enumeration scheme is illustrated in Figure 1 where our modification of the algorithm is incorporated.

Given a partial solution S, we can often detect some of the following three "S-conditions" (these conditions are not mutually exclusive):

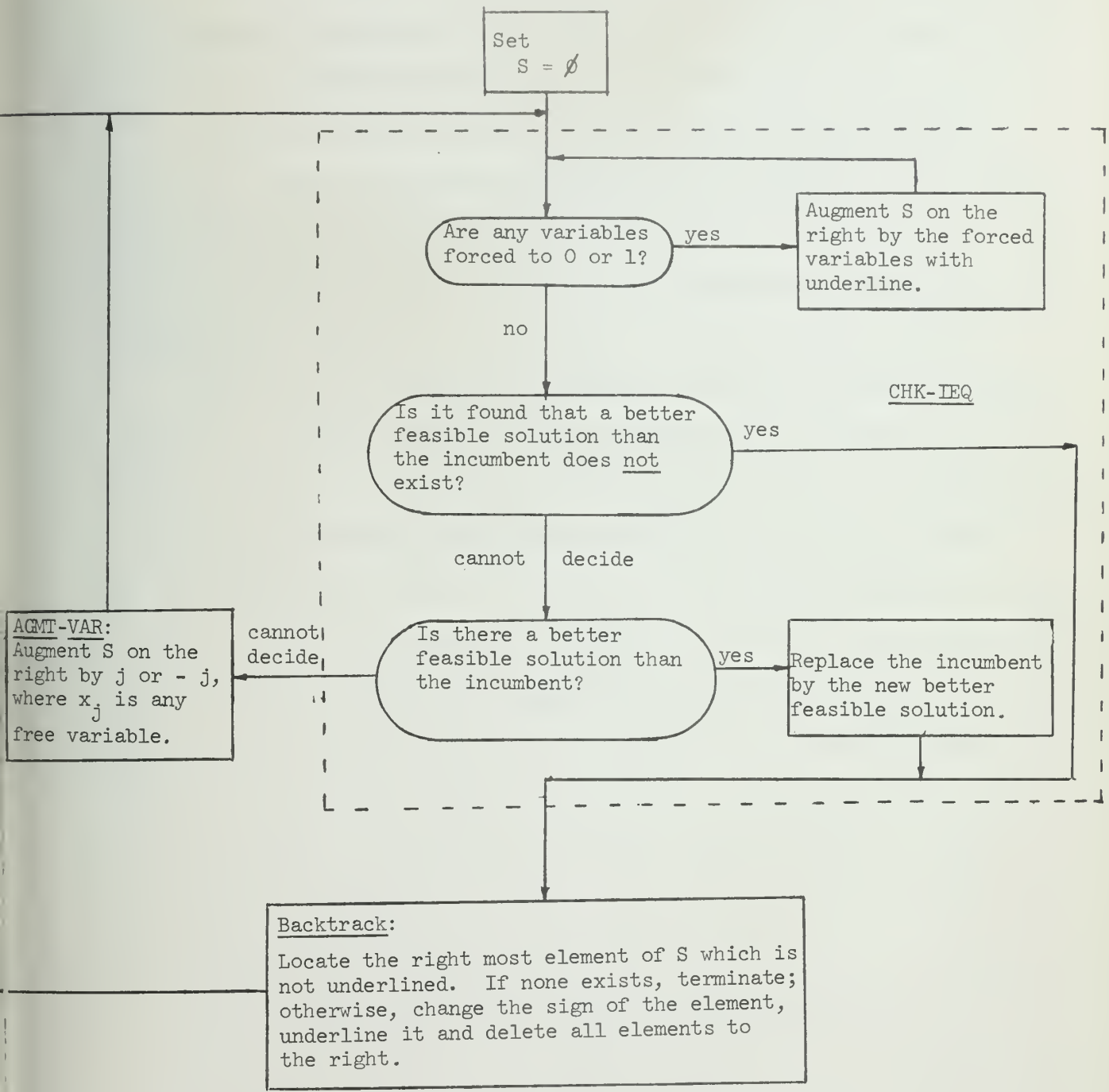
(a) S forces free variables to be specified in order to obtain feasible completions of the present partial solutions S,

(b) there is no better feasible completion of S, or even if there are some feasible completions, there's no better (smaller $\vec{c}\cdot\vec{x}$ than the incumbent) feasible completion of S, or,

(c) the best feasible completion of S is found and it is better than the incumbent.

If S forces some variables to be specified, we augment S by the specified variables with underline. For example if S = {3, 5, -2} forces x_1 = 0, the next partial solution is S = {3, 5, -2, $-\underline{1}$ }. In other words, an underline is attatched when the other partial solution which consists of the present partial solution and the opposite value of the specified variable (in the above example x_1 = 1) need not be checked for some reason, or it has already been examined.

Fig. 1 Implicit Enumeration Algorithm



When it is discovered that all feasible completions of S, if any, are worse than the incumbent, we simply backtrack. However, if the best completion of S is found and it is better than the incumbent, we replace the incumbent with this solution and then go to backtracking.

The backtracking scheme mentioned above is the same as that of Glover's[10] and is used because of its low storage requirement. (A detailed description of backtracking is found in references [10], [8], A rough idea may be seen in Figure 1.)

If with a given S we cannot detect any of the three "S-conditions", then we must pick one of the free variables and assign 1 or 0 to this variable, augmenting S on the right without underline (AGMT-VAR). Then go to CHK-IEQ.

As easily seen, the rate of convergence of implicit enumeration is greatly affected by the method of detecting the three "S-conditions" for a given partial solution S, and by the way of picking a free variable when no S-condition is detected. These will be discussed in some detail in the next section.

III DETECTION OF "S - CONDITIONS"

We will discuss our modified detection method of "S - conditions".

III-1 (a) and (b) of the S - Conditions (i.e. variable setting and no better feasible completion).

In checking each constraint we may be forced to set some variables to 0 or 1 in order to avoid infeasible completion of the current S. This same check may also reveal the infeasibility of a constraint or that there is no better feasible solution than the incumbent and therefore lead to backtracking.

ILLIP includes many of the techniques described in the literature [1], [6], [10], [15], [21]. However, some additional techniques which greatly increase the computational speed have been added to our algorithm.

To facilitate our computation, the objective function is converted to an inequality

$$\bar{z} - 1 - \vec{c} \cdot \vec{x} \geq 0 \qquad (3.1)^*$$

where $\bar{z} = \vec{c} \cdot \vec{x}^1$ and $\vec{x}^1$ is the incumbent solution. This inequality precludes all solutions which have the objective value larger than or equal to $\bar{z}$. The initial value of $\bar{z}$ is set to a known upper bound on the objective function. Each time we encounter a better feasible solution and replace the incumbent, we also replace the value of $\bar{z}$ of (3.1) by the objective function value of the new best solution.

If all optimal solutions are desired, replace (3.1) by

$$\bar{z} - \vec{c}\,\vec{x} \geq 0 \qquad (3.2)$$

and print all optimal solutions. Inequality (3.2) permits solutions with the same $\vec{c} \cdot \vec{x}$ value as the incumbent, thereby permitting multiple solutions.

---

* The constant - 1 in (3.1) is obtained by assuming all c_i to be integers.

Let (3.1) (or (3.2)) be rewritten as

$$b_0 + a_{01} x_1 + a_{02} x_2 + \cdots + a_{0n} x_n \geq 0 . \tag{3.3}$$

(2.1) with (3.3) together is denoted as

$$\vec{b}^* + A^* \vec{x} \geq 0 , \tag{3.4}$$

where $\vec{b}^*$ is an m + 1 dimensional column vector and A* is the $(m+1)\times n$ coefficient matrix. The objective of the implicit enumeration method is merely to find a feasible solution of (3.4) with the dynamic value of $\bar{z}$ (i.e., b_0 of (3.3)) properly assigned. This change allows us to limit the following discussion to only the feasibility of each inequality.

In order to facilitate this explanation the following example problem will be carried along.

$$\begin{aligned} &\text{minimize} && x_1 + x_2 \\ &\text{subject to} && \left.\begin{aligned} 1 - x_1 - x_2 + 2x_4 &\geq 0 \\ -3 + x_1 - x_2 + 3x_3 &\geq 0 \\ 1 - 2x_1 + x_2 - x_3 &\geq 0 \\ -x_1 - x_2 + x_3 - x_4 &\geq 0 \end{aligned}\right\} \end{aligned} \tag{3.5}$$

Therefore,

$$\vec{b}^* = \begin{bmatrix} 2 \\ 1 \\ -3 \\ 1 \\ 0 \end{bmatrix} \qquad A^* = \begin{bmatrix} -1 & -1 & 0 & 0 \\ -1 & -1 & 0 & 2 \\ 1 & -1 & 3 & 0 \\ -2 & 1 & -1 & 0 \\ -1 & -1 & 1 & -1 \end{bmatrix} \begin{matrix} \text{row number} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4, \end{matrix} \tag{3.6}$$

where $\bar{z}$ is initially set to 3. If

$$S = \{1, -2\}, \tag{3.7}$$

then a new set W which is a list of the variables occuring in S is

$$W = \{ 1, 2\}. \tag{3.8}$$

Now for any partial solution S we define three m+1 diminsional column vectors $\vec{y}(S)$, $\vec{\ell}(S)$, $\vec{u}(S)$ whose coordinates are

$$y_i(S) = b_i + \sum_{j \in W} a_{ij} x_j \tag{3.9}$$

$$\ell_i(S) = b_i + \sum_{j \in W} a_{ij} x_j + \sum_{\substack{j \notin W \\ a_{ij} < 0}} a_{ij} = y_i(S) + \sum_{\substack{j \notin W \\ a_{ij} < 0}} a_{ij} \tag{3.10}$$

$$u_i(S) = b_i + \sum_{j \in W} a_{ij} x_j + \sum_{\substack{j \notin W \\ a_{ij} > 0}} a_{ij} = y_i(S) + \sum_{\substack{j \notin W \\ a_{ij} > 0}} a_{ij}, \tag{3.11}$$

where W denotes the set of indries of assigned variables of S. In other words $y_i(S)$ is the current <u>value</u> of the i-th inequality summed over only the assigned variables of S. The value of $\ell_i(S)$ gives the <u>minimum value</u> for the i-th inequality since it is the current value plus all the negative coefficients of the free variables. Similiarly $u_i(S)$ is the <u>maximum value</u> on the i-th inequality for the partial solution S.

The following conditions justify the need for y_i, ℓ_i, u_i*:

(1) If $u_i < 0$ for some i, then the i-th inequality cannot be satisfied for any completion of the partial solution S. The partial solution S is infeasible and we backtrack to consider some other partial solution. This condition is seen in the third inequality in the example of (3.5) for S of (3.7), since $u_3 = -1$.

---

* The S from $y_i(S)$, $\ell_i(S)$, $u_i(S)$ will be dropped for notational clarity.

(2) If $|a_{ij}| > u_i$ and x_j is free, then

$$x_j = 0 \text{ if } a_{ij} < 0$$
$$x_j = 1 \text{ if } a_{ij} > 0 \qquad (3.12)$$

must hold. It is easily seen that these conditions hold since setting x_j to the opposite value will cause u_i to become negative and therefore infeasible. The special case $u_i = 0$ is particularly powerful since all free variables with non-zero coefficient will be set according to (3.12).

Again in the example of (3.5) u_2=1 and a_{23}=3. Thus x_3 can only be set to 1 with underline. Also, since $u_4 = 0$, both x_3 and x_4 must be set to 1 and 0, respectively in order to satisfy the 4 th inequality

(3) If $\ell_i \geq 0$ for some i, the i-th inequality can never assume a negative value for any completion of S and the inequality is not restrictive. In other words the i-th inequality can neither cause backtracking nor force any variables to be specified since it is always satisfied. Therefore we can ignore the i-th inequality until backtracking eventually discards the present partial solution S (i.e. some variables of S are eliminated or changed after backtracking).

In the example of (3.5) $\ell_1 = 0$ and regardless of the values assigned to x_3 and x_4 the first inequality is always satisfied for S of (3.7).

In other words, the detection of the S-conditions (a) and (b) can be done only through the above properties (1), (2) and (3). If other effective checking procedures are available they could also be incorporated without changing the basic scheme of enumeration. However we adopted properties (1) and (2) to check the S-condition (a) and (b) because they seem simple and efficient enough for our problems. Then several gimmicks are added to facilitate the computation further, as will be explained hereafter.

The values of column vectors $\vec{\ell}$, $\vec{u}$, $\vec{y}$ are adjusted whenever a variable is attached or deleted from the partial solution S rather than calculating each entry every time they are needed. In this way the amount of computation is greatly reduced.

If $|a_{ij}| > u_i > 0$, we need at least $|a_{ij}| \geq 2$* for the condition to work. Hence it is helpful to computation, to have a column vector $\vec{d}$ which indicates the existence of those coefficients whose absolute value is larger than 1 in each inequality. This is especially effective when the majority of the non-zero coefficients are 1 or -1.

In our problem of optimal logic network synthesis, all non-zero coefficients in most inequalities had the absolute value of 1 except one coefficient. (Some inequalities may have more than one coefficient whose value is smaller than -1 or larger than +1.) In this problem $\vec{d}$ was created according to the following scheme:

$$d_i = \begin{cases} 0 & \text{if } |a_{ij}| \leq 1 \text{ for all } j = 1, 2, \ldots, n, \\ j_0 & \text{if } |a_{ij}| > 1 \text{ for } j = j_0 \text{ and } |a_{ij}| \leq 1 \text{ for } j \neq j_0, \\ * & \text{otherwise.} \end{cases}$$

With the aid of these $m + 1$ d_i values we can save much searching and testing to find $|a_{ij_0}| > u_i > 0$.

Another new scheme adopted in our program is the "cyclic checking" of constraints. By this we mean that the inequality check proceeds from row 0 to row m and repeats 0 to m over and over again until no more changes occurs, i.e. non of the three S-conditions occurs.

---

* Assumes all a_{ij} are integer.

This scheme leads to more variables being set and consequently the preclusion of more solutions in our enumeration.

In addition to the columns $\vec{\ell}$, $\vec{u}$, $\vec{y}$, $\vec{d}$ discussed above, introduce a new $(m + 1)$ - dimensional column $\vec{t}$ which works as a tag for each inequality indicating whether it should be checked or not, in order to facilitate the above cyclic checking. If $t_i = 1$, the i-th inequality should be checked to determine if some S-conditions hold. If $t_i = 0$, the check for the i-th inequality is skipped.

We set the initial value of $\vec{t}$ and update its value as follows:

(a) Initial value is:

$$t_i = 1 \qquad \text{if} \qquad \ell_i < 0$$

$$t_i = 0 \qquad \text{if} \qquad \ell_i \geq 0,$$

for $i = 0, 1, \ldots, m$,

because $\ell_i > 0$ demonstrates that the i-th inequality is non-restrictive and has absolutely no influence on any completion of S.

(b) After checking the i-th inequality, set $t_i = 0$ since there is no need to check this inequality again unless t_i is set back to 1 by the next condition.

(c) Whenever a free variable x_j is specified, the vector $\vec{t}$ must be updated.

(c.1) If $x_j = 0$, then

$$\begin{cases} t_i(S') = 1 & \text{if } a_{ij} > 0 \quad \text{and } \ell_i(S') < 0, \\ t_i(S') = 0 & \text{if } a_{ij} < 0 \quad \text{and } \ell_i(S') \geq 0, \\ \text{no change} & \text{otherwise,} \end{cases} \tag{3.13}$$

where the new partial solution S' is S with $x_j = 0$ added.

To prove this, first recall that the check of (a) and (b) of the S - conditions is done only by checking whether $\ell_i \geq 0$, $u_i < 0$ and $|a_{ij}| > u_i$ hold (see the properties (1), (2) and (3).). Thus the value of t_i should be reconsidered only when those values have been changed by setting a certain variable. Next note that if $a_{ij} < 0$ and x_j is set to 0, then $\ell_i(S') > \ell_i(S)$ but $u_i(S') = u_i(S)$, and if $a_{ij} > 0$ and $x_j = 0$, then $u_i(S') < u_i(S)$ but $\ell_i(S') = \ell_i(S)$. The first case of (3.13) is true since by the new smaller value of u_i, some free variables may be forced to 0 or 1 in this case. The second case of (3.13) must be added because of $\ell_i(S') \geq 0$ by setting x_j to 0 thereby eliminating the necessity of checking the i th inequality. Other two cases:

$$a_{ij} > 0 \text{ and } \ell_i(S') \geq 0 ,$$

$$\text{and} \quad a_{ij} < 0 \text{ and } \ell_i(S') < 0$$

cause no change as far as the S - conditions are concerned and consequently the tag t_i is not changed. (In the first case, t_i needs not be changed since if $\ell_i(S') \geq 0$, then $\ell_i(S) \geq 0$. In the second case, t_i needs not be changed since the value of u_i does not change.)

(c.2) If $x_j = 1$, then

$$\begin{cases} t_i(S'') = 1 \text{ if } a_{ij} < 0 \text{ and } \ell_i(S'') < 0, \\ t_i(S'') = 0 \text{ if } a_{ij} > 0 \text{ and } \ell_i(S'') \geq 0, \\ \text{no change otherwise,} \end{cases} \tag{3.14}$$

where S'' is S with $x_j = 1$ added.

The proof is similiar to that just described for the $x_j = 0$ case.

Initially many of the t_i's are 1. Starting with t_0 we check in a round-robin fashion each i th inequality for which $t_i = 1$.

We continue this procedure, each time specifying a variable if it is forced, until:

(A) some $u_i < 0$ which shows that the present partial solution S is infeasible and we go to the backtracking procedure,
(B) A feasible solution is generated, or
(C) All the $t_i = 0$ which shows that no further information can be obtained from $\vec{u}, \vec{\ell}$ and $\vec{y}$ at this moment.

If the case (C) occurs, the partial solution will be augmented by some variable without underline (see Fig. 1). In this case which variables is augmented can significantly affect the convergence speed. A few different methods for picking this variable are discussed in section IV.

When we have backtracking, some variables are deleted from the partial solution S and $\vec{t}$ must be altered again. Let us consider the updating procedure of $\vec{t}$ by an example. Assume that we had the backtracking procedure with the partial solution

$$S = \{ -\underline{1}, -\underline{3}, 2, \underline{5}, \underline{7}, -\underline{4}\}. \tag{3.15}$$

After underlining x_2, changing its sign and discarding the suceeding variables in the backtracking procedure, S becomes

$$S' = \{ -\underline{1}, -\underline{3}, -\underline{2} \}. \tag{3.16}$$

$\vec{t}(S')$ could be calculated from $\vec{t}\,(S)$ according to the above procedure. However the following procedure was actually taken, because of easy calculation.

Because $x_2 = 1$ was included in S of (3.15) without underline and augmentation of a new variable without underline results from the above case (C) only, we must have had

$$S'' = \{ -\underline{1}, -\underline{3}\}$$

before S and also $\vec{t}(S'') = \vec{0}$ must hold for this S". Therefore we can

calculate $\vec{t}(S')$ directly from $\vec{t}\,(S'')$ since S' is S" with $x_2 = 0$ added.

As seen from the above example, the general rule is as follows; After locating the rightmost variable x_j of the partial solution S, which is not underlined, the new $\vec{t}\,(S')$ is calculated from $\vec{t}\,(S'')$ of the following properties,

(i) $\vec{t}\,(S'') = 0$

(ii) S' is the new partial solution which is S" augmented with the opposite value of x_j.

Consequently the computation of t is fairly simple and can easily be implemented.

Another gimmick to be mentioned is that in order to utilize the low density of non-zero coefficients, only non-zero coefficients are packed in the computer memory in two ways: one is columnwise, i.e., the non-zero coefficients in a column are stored consecutively so that information for each column may be read out efficiently, and the other is rowwise. The more convenient of these two data are used in each of actual procedures.

III-2 (c) of the S - conditions (i.e. detection of feasible solution.)

During the course of the algorithm, the best feasible completion of S, in the sense that it attains the minimum objective function value among all feasible completions of S hitherto obtained can be often calculated. It can be calculated by any one of the following three approaches:

(a) Detection of $\vec{y}(S) \geq 0$.

From the definition of $\vec{y}$ in equation (3.9), this condition indicates that the completion of the present partial solution S obtained by specifying all free variables to 0 is feasible and moreover it is the best completion of S since all coefficients c_i of the objective function are non-negative. This case was first mentioned by Balas [1].

b) Detection of $\vec{\ell}(S) \geq 0$.

This condition shows that any completion of S is feasible. As a result, the best completion is obtained by specifying all free variables to 0.

c) A paritial solution with no free variable left.

In this case, since all variables are specified, there is only one completion of S and that is S itself which is obviously the best.

Whenever the best feasible completion of the current partial solutio S is found by any of the above methods, it replaces the incumbent. (Because of (3.1) a feasible solution always gives a better objective value). The backtracking procedure is then initiated.

Obviously the decision of which test of the above (a), (b) and (c) is used should be done according to the nature of the given problem. In the excution reported hereafter, however, (c) is used throughout because of a computational simplicity.

When <u>all</u> optimal feasible solutions are desired an enumeration procedure is not as simple as the case of a single optimal feasible solution. A case when all the coefficients c_i of the objective function are positive can be also dealt with by the same discussion as the previous one (inequality (3.2)). If some of c_i are 0, however, we need a more careful consideration. Each of the above tests (a), (b) and (c) needs the following modification. If the $\vec{\ell} \geq 0$ test is used, the set of completions of S obtained by specifying each free variable by

$$\begin{aligned} x_i &= 0 \quad , \text{ if } c_i > 0 \\ x_i &= 1 \text{ and } 0 \, , \text{ if } c_i = 0. \end{aligned} \tag{3.17}$$

(a) can detect at the earliest chance, (b) is the next and (c) is the last. But the incorporation of (a) or (b) is more complex than (c).

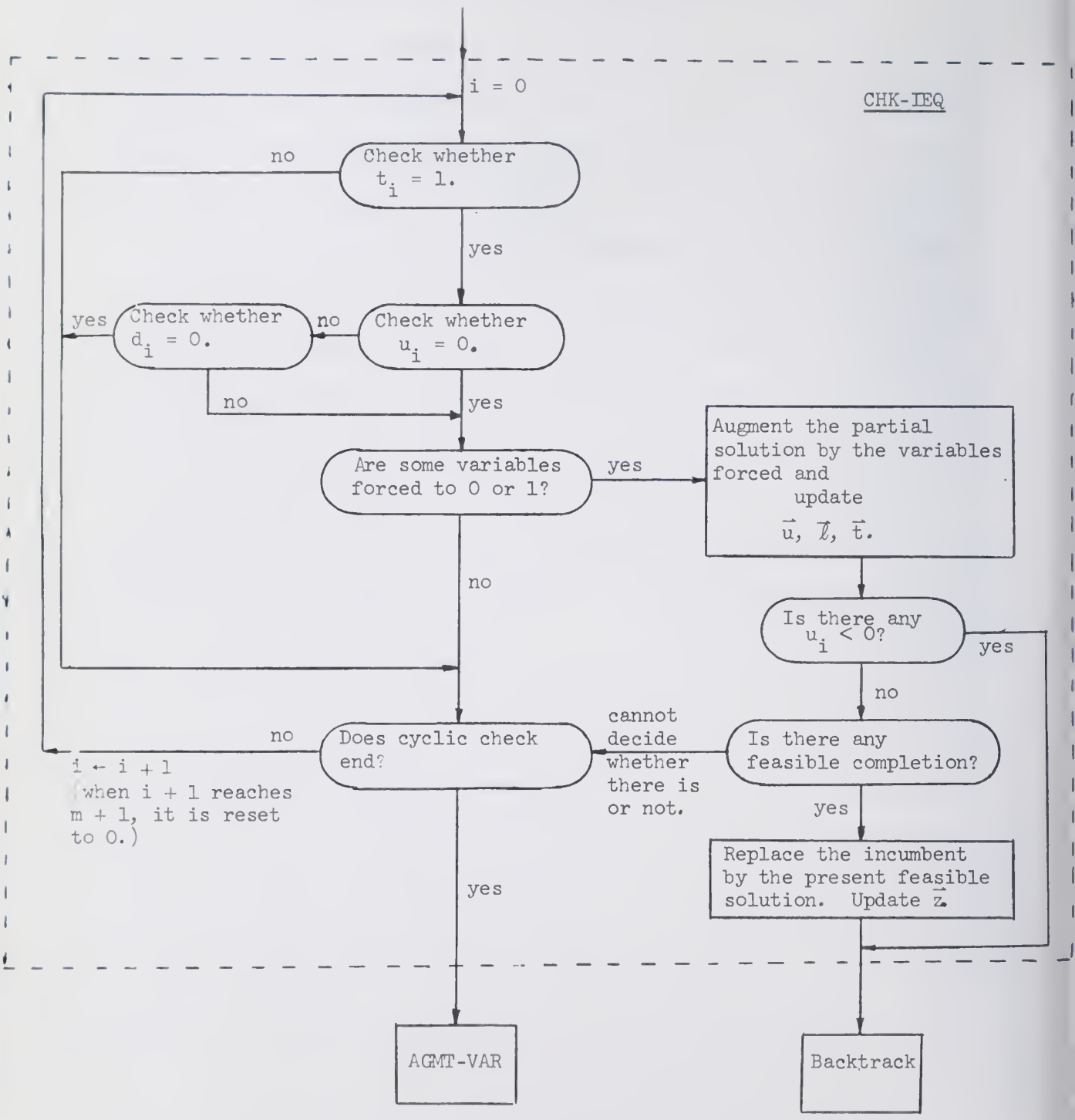
exhausts all the best feasible completions of S. By using (3.17), it is easy to generate all optimal solutions.

On the other hand, if $\vec{y} \geq 0$ test is used, a more involved argument is needed . First, consider the set of completions of S discussed in (3.17). In this case, however, some of completions may not be feasible. Therefore, by checking the feasibility of every completion of S obtained by assigning free variables according to (3.17) we can get the set of all the best feasible completions of S.

When all the free variables of a paritial solution are forced to 1 or 0, the feasible completion of S is uniquely obtained, regardless of the value of the coefficients in the objective function.

The procedure for checking inequalities including the "cyclic check" and the procedure for detecting feasible solutions are summarized in the flow diagram of Figure 2. It is named CHK-IEQ. The flow diagram should be self-explanatory.

Figure 2 Flow chart of procedures in CHK-IEQ



IV Augmentation of a Variable to Partial Solution

When no further information can be obtained from checking S-conditions, the current partial solution is augmented by a variable without underline. The finite convergence of algorithm is guaranteed whichever free variable is chosen, but it heavily affects the convergence speed and therefore should be carefully selected.

So far two approaches for this selection have been favored and proved to be efficient by computational results.[6][7][21] One is to attempt to find a feasible solution as soon as possible without considering the objective value[1]. The other is to attempt to proceed to a better feasible solution possibly at the sacrifice of prompt discovery of a feasible solution.[10][21] In the latter approach, once a good feasible solution is stored, however, it will significantly facilitate the rest of the enumeration.

This part of algorithm is denoted as AGMT-VAR and four methods implemented by us are now presented. Computational results of these methods of AGMT-VAR show a very irregular computation time. No method is better than others for all problems.

(1) Balas' method[1]*: Let the current partial solution be S. Calculate

$$\sum_{i=0}^{m} \min \{ y_i(S) + a_{ij}, 0 \} \tag{4.1}$$

for each free variable x_j and pick up the subscript j_0 which maximizes (4.1). Then the next partial solution is

$$S' = \{S, j_0\}. \tag{4.2}$$

---

* This is slightly different from the original.

(2) For each free variable x_j, let L_j be the number of inequalities which satisfy both

$$\ell_i(S) < 0$$
$$\text{and} \quad \ell_i(S) + a_{ij} \geq 0. \tag{4.3}$$

Pick up j_0 which maximizes L_j. The next partial solution is

$$S' = \{S, j_0\}. \tag{4.4}$$

In other words, this method is aimed at maximizing the number of inequalities with $\ell_i(S') \geq 0$ when 1 is assigned to x_j, so that the new partial solution may possess as many non-restrictive inequalities as possible. If several variables have the same number of inequalities with $\ell_i(S') \geq 0$, a secondary criterion may be applied to break the tie. A criterion which we used was as follows: among j's which attain the maximum L_j, calculate the value

$$\sum_i a_{ij}, \tag{4.5}$$

for all i's such that

$$a_{ij} > 0$$
$$\ell_i(S) + a_{ij} < 0. \tag{4.6}$$

Pick up the j_0 which maximizes (4.5) and the next solution is (4.4) in the above.

(3) While (2) was derived under the assumption that x_j would be specified to 1 only, (j_0 has no minus sign in (4.4)) the third method is a generalization of (2) by allowing each variable to be specified to either 0 or 1. Let the number of inequalities which satisfies (4.3) be L_j^1 for a free variable x_j. L_j^0 is defined as the number of inequalities which satisfy

$$\ell_i(S) < 0$$
$$\text{and} \quad \ell_i(S) - a_{ij} \geq 0, \tag{4.7}$$

for each free variable x_j. Suppose that $L_{j_0}^{\epsilon}$ is the maximum among L_j^1 and L_j^0 ($\epsilon = 0$ or 1), then the next partial solution is

$$S' = \{ S, (-1)^{1-\epsilon} j_0 \}. \tag{4.8}$$

The similar secondary criterion as (2) may also be incorporated to break a tie.

(4) The fourth method includes a consideration on the objective value. Although various sophistricated methods which are based on the objective value are now available[10][21], our fourth method uses the number L_j^1 defined as in the above. Then calculate

$$L_j^1 / (\alpha c_j + \beta) \tag{4.9}$$

for each free variable, where α an β are constants.* Let j_0 attain the maximum of (4.9). The next partial solution is

$$S' = \{S, j_0\}.$$

---

* In our computation α and β were set to 1.

This is aimed at leading to a partial solution closer to feasible solutions hopefully with the minimum increase of the objective value.

There have been a number of other methods available so far and a variety of modifications are also conceivable. However, rigorous theoretical comparison of these appears to be difficult since the efficiency of each approach heavily depends on each individual problem. As shown in Section VI, a specially tailored algoritham for each given problem might be necessary to attain the maximum efficiency. However, the above approachs are believed to be reasonably efficient for general problems in spite of its simplicity.

V Underline and Pseudo-Underline

This section will discuss a new condition of underlining and then the concept of pseudo-underlining in a partial solution which were probably not discussed in the literature in the past.

In our implicit enumeration, if many inequalities become $\ell_i \geq 0$, showing that they are not restrictive, the remaining inequalities (i.e., inequalities with $\ell_i < 0$) may have chances to satisfy the following condition for some free variable x_j:

$$(1) \quad a_{ij} \leq 0 \quad \text{for all i for which } \ell_i(S) < 0 \tag{5.1}$$

$$(2) \quad a_{ij} \geq 0 \quad \text{for all i for which } \ell_i(S) < 0\,, \tag{5.2}$$

where S is the current partial solution.

If case (1) occurs, $x_j = 0$ can be immediately added with underline to the current partial solution. Namely, the next partial solution is

$$S' = \{S, \underline{-j}\}. \tag{5.3}$$

This is because, under the condition (5.1), if a feasible completion of S is obtained by specifying x_j to 1 and other free variables to appropriate values, the completion with only x_j changed to 0 (other free variables keep the same values) is also feasible. Moreover, from the condition that $c_i \geq 0$, the objective value for the latter completion is not worse than the former. Therefore we can preclude all the completion with $x_j = 1$ from the further enumeration*. This is equivalent to considering the next partial solution as (5.3).

On the other hand, if case (2) occurs, $x_j = 1$ can be augmented to the partial solution S with pseudo-underline which is denoted by "~". The next solution is thus denoted by

$$S' = \{S, \underset{\sim}{j}\}. \tag{5.4}$$

---

* It is assumed that a single optimum solution is seeked for. The procedure can be easily modified for the case of all optimum solutions, based on the modification discussed in Section III-2.

Pseudo-underline is defined as follows: if no feasible completion follows from S', a pseudo underline works as underline, but if a feasible completion is detected, pseudo underline is deleted and x_j is considered as a variable without underline.

A proof is given below. If the partial solution {S,j} induces no feasible completion, neither does {S,-j} under condition (5.2). Hence, in this case we can exclude the enumeration for {S,-j} and it implies that x_j can be added with underline. However, if some completions of {S,j} are feasible, some completions of {S,-j} might be feasible and may attain better objective values, thus prohibiting the addition of x_j with underline. Of course if $c_j = 0$, the pseudo underline of x_j can be treated as underline since {S,-j} can not attain a better feasible solution.

Backtracking procedure with the above two cases for underlining and pseudo-underlines is the same as that in Section II except that pseudo-underlines are deleted from the partial solution whenever a feasible solution is detected. The termination of enumeration would be speeded up by this gimmick especially when the given problem is infeasible.

The above two gimmicks for conditions (5.1) and (5.2) were incorporated in our program to test their effect on the computation speed. For many problems a reduction of the number of iterations was observed. Because of this improvement, the two gimmicks are incorporated in computer programs with which many problems are solved in a later section. In the actual implementation, condition (5.1) is checked for all free variables after finishing the cyclic check whereas condition (5.2) is checked for only variable which is selected by AGMT-VAR.

Although an idea behind the gimmick for condition (5.1) is the same as that of the set E_s defined by Balas[1], it does not seem to have been explicitely discussed in literature in the past that we can underline a variable subject to condition (1). The second gimmick for condition (5.2) probably has not been discussed in the past.

VI Computational Results and Specialization of the Algorithm

This section includes computational results applied to such problems as the optimum network synthesis problems for which our program was initially designed, the covering problems, the travelling salesman problems and several other problems found in the literature. The program was written in FORTRAN IV and run on IBM 360/75 I. As will be observed from the results, our program seems powerful for such problems as optimum network synthesis problems which have features mentioned in Section I. Probably it is because mutual relationship of variables is very tight in such problems and specifying one variable causes many other free variables accordingly specified. In other words, the S-conditions discussed in Section III work effectively, even though the check is quite simple.

However, for other problems our program is not always the fastest. For these problems, the S-conditions do not give information sufficient to preclude many partial solutions.

In this paper we try to make use of intrinsic properties of given problems such as the optimum network design problems, the covering problems and the traveling salesman problems, rather than adding new procedures which may be effective for all types of problems. As will be explained individually, the simple modifications result in the considerable improvement. This direction, however, is not throughly explored and it should be further done so elsewhere. The results shown hereafter is mainly to show the adaptability of the implicit enumeration algorithm. The modifications added for each problem are of the following types:

(1) New checking methods of the S-conditions other than those discussed in Section III.

(2) New AGMT-VAR algorithms based on the structure of given problems.

(3) The estimation of the bound of objective function obtainable from the current partial solution. If we can obtain a lower bound of objective function by some means during computation and if the lower bound exceeds the objective value of the incumbent, the current partial solution is discarded and the backtracking procedure immediately follows.

Modifications actually added will be explained later together with the given problems.

VI-1 Optimum Network Synthesis Problems

Table 6.1 shows the computational results of our algorithm applied to optimum network synthesis problems, for which it was initially designed. The optimum network synthesis is to find a logic network consisting of NOR gates which realizes a given switching function, and which has the minimum number of gates and, as the secondary condition, minimizes the number of interconnections. Integer programming approach to this problem has various advantages over others, such as the easy incorporation of various restrictions on a network to be synthesized, the applicability of the formulation to other types of gates and the choice of a wide variety of objective functions to be optimized. Detailed formulation and further consideration will be found in [19], [20], [3], together with syntheses of other types of networks.

Table 6.1 is the result of exhaustion of all optimum network for every three variable switching function. It is not only interesting as integer programming problems since considerably large problems were solved in reasonable computation time, but also it is a new approach to such optimum network synthesis problems. The percentage of non-zero elements is low (e.g. about 2% for R = 6) and decreases as R grows. In case of NOR gate network synthesis the integer programming approach seems more efficient than any other existing methods[13], if optimum networks are synthesized under general restrictions such as fan-in and fan-out restrictions or without restrictions on the number of levels.

R in Table 6.1 is the number of gates in the network to be synthesized. Therefore if we solve a integer program for R gate network and if the switching function to be realized actually needs more than R gates, the problem turns out to be infeasible. This infeasible case is also shown in

Table 6.1 Optimum NOR network synthesis problems
(No fan-ins and fan-outs restrictions)

Size			General approach (ILLIP)				Specialized approach	
			Feasible		Infeasible		Feasible	
Number of gates R	No. of variables	No. of inequalities	Average computation time** for each function. (sec)	Average number of iterations* for each function.	Average computation time** for each function. (sec)	Average number of iterations* for each function.	Average computation time** for each function. (sec)	Average number of iterations* for each function.
3	52	88	0.28	4	0.08	2	----	---
4	90	169	0.90	30	0.49	27	----	---
5	137	265	3.59	104	1.99	73	----	---
6	193	415	42.26	954	30.52	843	----	---
6	193	423	----	---	-----	---	4.99	136
7	258	716	982	15,908	-----	---	----	---
8	395	1012	----	----	-----	---	390.0	4822

* The number of iterations means the number of visits to CHK-IEQ during the computation.

** The machine used for computation is the IBM 360/75I.

Table 6.1. It may be worth mentioning that an infeasible problem usually needs less computation time than a feasible problem of the same size.

The algorithm was further improved by modifying AGMT-VAR in order to take into consideration the inherent property of NOR gate network[3]. Also a strict lower bound estimation of the objective function based on the network structure is incorporated. The result is shown in the column "Specialized approach" in Table 6.1 for R = 6. It is considerably faster than the result for R = 6 in the column "General approach".

The optimum networks for the switching function that needs 8 NOR gates under fan-ins and fan-outs restrictions were solved for the first time by this approach.

For these problems, Gomory's algorithm[11] was also applied, but was not efficient.[2]

VI-2 Traveling Salesmen Problems

Traveling salesman problems were also solved by the implicit enumeration algorithm. Although the integer programming approach seems inferior to other algorithms well tailored to the problem, such as "branch and bound" method according to the computational experience,[4] it is attempted because the traveling salesman problem is a typical integer programming problem. As will be seen, a simple modification of the algorithm to utilize inherent properties of the problem works remarkably well.

The integer linear formulation used is that by Miller, Tucker and Zemlin[18], in which a variables x_{ij} is associated with the path from the i th city to the j th city, i, j=1, 2, ..., N, $i \neq j$, and a variable* u_i with the i th city, i = 1, 2, ..., N. N is the number of cities to be considered. Let d_{ij} be the distance from the i th city to the j th city, then the N city traveling salesman problem is formulated by;

$$\text{minimize} \qquad \sum_{i \neq j} \sum d_{ij} x_{ij} \tag{6.1}$$

$$\text{subject to} \qquad \sum_{\substack{i=1 \\ i \neq j}}^{N} x_{ij} = 1 \qquad j = 1, 2, \ldots, N \tag{6.2}$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{N} x_{ij} = 1 \qquad i = 1, 2, \ldots, N$$

$$u_i - u_j + (N-1)\, x_{ij} \leq N-2 \tag{6.3}$$

$$2 \leq i \neq j \leq N,$$

* See [18] for what u_i represents.

where x_{ij} is a zero-one variable but u_i may assume an integral value up to N-2. Each u_i is expanded with zero-one variables u_{ik}'s as

$$u_i = \sum_{k=1}^{N-2} u_{ik}, \tag{6.4}$$

rather than using the binary number expansion of u_i for the reason which will be explained later.

First, the general approach with method (2) of Section IV (this will be abbreviated as AGMT-VAR 2) was attempted and the result is shown in Table 6.2.

To improve this result, the following three modifications are made. The first modification is to exclude variables u_{ik} from the enumeration procedure because these variables are only for eliminating subtours which do not pass through the first city. In other words, variables u_{ik} are added to a partial solution always with underline and therefore none of them will be choosen for underling in the backtracking procedure. The procedure will be given in the following. The justification of the procedure is not given, but it should be noticed that the expansion (6.4) is necessary for this purpose.

Let $\vec{y}_A$ be the coordinates of $\vec{y}$ defined by (3.9) which correspond to the set of inequalities (6.3). Then if $\vec{y}_A \geq 0$ does not hold, free variables which have coefficient 1 in the inequalities with negative value of $\vec{y}$, are successively specified to 1 with underline until the value of $\vec{y}_A$ becomes non-negative. If this is not posssible, it means the existence of a subtour and the backtrack is initiated.

The second modification is a new AGMT-VAR procedure which picks up a variable among free variables x_{ij}'s with the minimum value of a_{ij}. This variable is of course augmented to the partial solution without underline.

Number of Cities*	Size		With AGMT-VAR 2		Specialized algorithm		Other algorithm published in the past		
	Variables	Inequalities	Time** (sec)	Iterations	Time** (sec)	Iterations	Time (sec)	Machine	Ref.
6 (Symmetric)	50	44	14.61	1,665	0.62	83			
6 (Asymmetric)	50	44	14.26	3,447	0.13	16			
6	35	31					51.6	IBM 7094	[5]
10 (Symmetric)	162	112	>360.0	> 25,000	102.56	6073			
10 (Asymmetric)	162	112	>350.0	> 25,000	2.60	156			
12 (Asymmetric)	242	158	------	------	58.16	2782			

Table 6.2 Computational results with Traveling Salesman Problems

* When $d_{ij} = d_{ji}$ holds, the distance is called symmetric.
otherwise it is called asymmetric.

** The machine used for computation is the IBM 360/75 I.

The third one is the new bound estimation. First, let B be the set of cities for which no incoming path is specified yet. For each $j \in B$, find

$$d_j = \min_i d_{ij} \tag{6.5}$$

where i runs over all i corresponding to free variables x_{ij}. Then set

$$D_1 = \sum_{j \in B} d_j. \tag{6.6}$$

Similarly, C is the set of cities for which no outgoing path is specified yet. For each $i \in C$, find

$$d_i = \min_j d_{ij}, \tag{6.7}$$

where j runs over all j corresponding to free variables x_{ij}. Set

$$D_2 = \sum_{i \in C} d_i \tag{6.8}$$

Let D be the larger of D_1 and D_2. Obviously D plus the objective value of the current partial solution can be used as a lower bound of the objective value obtainable from the current partial solution.

With these modifications added, the computation time was significantly improved as shown in the column "Specialized algorithm" in Table 6.2, though the result is still much inferior to other reported algorithms such as branch and bound methods.[4] The result is demonstrated only for the purpose of showing that an augmentation of the algorithm derived from inherent properties of the problem leads to a significant improvement in spite of its simplicity.

VI-3 Set Covering Problems

A set covering problem is;

$$\text{minimize} \quad \sum_{i=1}^{n} x_i$$

$$\text{subject to} \quad A\vec{x} \geq \vec{1} \tag{6.9}$$

where $\vec{x}$ is an n-dimensional vector of zero-one variables, A is an $m \times n$ matrix consisting of only 0 and 1, and $\vec{1} = (1, \overset{m}{\ldots}, 1)^T$. Matrix A was randomly generated with the density 0.07 of coefficients 1. m = 30 in all problems.

Five samples for different n were solved and the average solution time is shown in Table 6.3.

A specialized algorithm is the algorithm with AGMT-VAR 2 modified in the following two respects:

(1) A new checking method of partial solution S.

Let $\vec{a}_i$ be the i th column of A and $\vec{a}_i(S)$ be the $\vec{a}_i$ from which entries corresponding to all rows k such that $\ell_k(S) \geq 0$ are deleted. Namely $\vec{a}_i(S)$ is composed of inequalities which are restrictive under the partial solution S. Then if

$$\vec{a}_i(S) \leq \vec{a}_j(S) \tag{6.10}$$

holds for at least one j such that $j \neq i$ and $j \in W$, then x_i is specified to 0 with underline.

(2) An objective bound estimation.

Let E(S) be the number of restrictive inequalities under the partial solution S, i.e. the number of inequalities with $\ell_i(S) < 0$. Corresponding to a free variable x_j,

With AGMT-VAR 2			Specialized algorithm		Geoffrion's algorithm	
Number of Variables* (n)	Time (sec)	Iterations	Time on IBM 360/75I (sec)	Iterations	Time on IBM 7044 (sec)	Iterations
30	0.13	35	0.03	1	1.8	3
40	1.92	563	0.11	3	4.2	6
50	13.05	3714	0.15	5	4.8	4
60	66.57	13844	0.37	12	8.4	6
70	---	---	0.38	12	9.0	8
80	---	---	0.93	40	12.6	6
90	---	---	1.60	72	10.2	4

Table 6.3 Computational results with set-covering problems

* Each problem has exactly 30 inequalities.

$E_j(S)$ is the number of coefficients a_{ij} in the j th column with $a_{ij} = 1$ and at the same time $\ell_i(S) < 0$. Let us order $E_j(S)$'s in such a way as

$$E_{j_1}(S) \geq E_{j_2}(S) \geq \cdots\cdots , \tag{6.11}$$

and define k be the smallest number which satisfies

$$\sum_{p=1}^{k} E_{j_p}(S) \geq E(S). \tag{6.12}*$$

Then the objective value corresponding to variables in S and k can be used as a lower bound of objective function obtainable from S.

Adding these two modifications, the result is shown also in Table 6.3. The set-covering problem has been extensively studied in conjunction with the minimization of prime implicant table discussed in the switching theory[17][5]. Various techniques known in the area could also be applied to improve the computation further, though we did not try yet.

Table 6.3 shows the result of algorithm applied by Geoffrion[9] to similar problems in which he incorporated the linear programming as an aid. His result is impressively good and the increase of computation time seems linearly proportional to the number of variables.

---

* Any partial solution S for which (6.12) does not hold for any k can be always found to make inequalities infeasible, by checking the sign of u_i.

VI-4 Problems Taken From Literature

Table 6.4 shows computational results for various problems* taken from literature[12][15]. AGMT-VAR which among those in Section IV gave the best performance for each problem is also shown.

Compared with other algorithms, our algorithm does not always give the best performance. However, as expected, it works very well for such problem as IBM 9 which has more number of inequalities than the number of variables (this characteristic is similar to the optimum network synthesis problem.)

For general use, our approach may need modifications such as the incorporation of linear programming as suggested by Geoffrion[9], since his computational results seem quite efficient constantly for various problems as seen from Table 6.4. However we believe that gimmicks discussed in earlier sections considerably speeded up the original implicit enumeration procedure of Balas[1], Glover[10], Geoffrion[8] and Fleischmann[6].

---

* Some of the problems actually solved were obtained from H. Salkin of the IBM. The authors are grateful for his generous help.

Problem	Size		Our Approach			Algorithm proposed by different authors		
	No. of variables	No. of inequalities	Time on IBM 360/75I (sec)	No. of iterations	The best AGMT-VAR among those in section IV.	Time (sec)	Machine	Ref.
Haldi [12]								
II - 1	14	4	0.09	40	2			
II - 7	20	4	1.07	382	3	1.8	IBM 7044	[9]
II - 8	20	4	1.48	546	3	3.0	7044	[9]
II - 9	15	6	0.08	32	3			
II - 10	30	10	3.56	757	1	3.6	7044	[9]
IBM [12]								
2	21	7	2.36	809	1	1.2	7044	[9]
6	31	31	> 120.	> 25,000	2	120.0	7044	[9]
9	15	35	1.68	489	1	28.4	7044	[9]
Lemke and Spielberg [6]								
A	12	6	0.03	15	2	60.0	IBM 360/40	[15]
C	44	12	11.26	2108	2	120.0	360/40	[15]
B	35	28	2.45	474	2	120.0	360/40	[15]
D	74	37						

Table 6.4 Computational results with problems taken from literature

CONCLUSION

ILLIP is implemented initially to solve problems encountered in the optimum logical network synthesis. Various new ideas are incorporated to further speed up the computation. As seen from the computational results, the program works particularly well for such problems as the optimum networks synthesis problems in which there are more number of inequalities than the number of variables and accordingly specifying a variable to 0 or 1 causes many other variables to be specified.

The programming manual of ILLIP is available through Liu[16].

In general, however, a further improvement of the algorithm may be necessary to obtain the maximum computation speed. The implicit enumeration algorithm is quite convenient for this purpose because the algorithm can be easily modified to a great extent by simply adding new checking procedures of inequalities in CHK-IEQ or by modifying AGMT-VAR. Some problem-oriented modifications were tried on the optimum network synthesis problems, the traveling salesman problems and the covering problems, resulting in the significant improvement. This direction should be further investigated in the future.

REFERENCES

(1) E. Balas, "An Additive Algorithm for Solving Linear Programming With Zero-One Variables," Operations Research, vol. 13, no. 4, pp. 517-544, July-August, 1965.

(2) C. R. Baugh, T. Ibaraki and S. Muroga, "Computational Experience in All-Integer, Binary Variable, Integer Programming Problems Using Gomory's All-Integer Algorithm," Report No. 259, Department of Computer Science, University of Illinois, April, 1968.

(3) C. R. Baugh, T. Ibaraki, T. K. Liu and S. Muroga, "Optimum Network Design Using NOR and NOR-AND Gates by Integer Programming," to be published.

(4) M. Bellmore and G. L. Nemhauser, "The Traveling Salesman Problem: A Survey," Operations Research, vol. 16, no. 3, pp. 548-558, May-June, 1968.

(5) A. Cobham, R. Fridshal and J. H. North, "An Application of Linear Programming to the Minimization of Boolean Functions", Proc. Second Ann. Symp. of Switching Circuit Theory and Logical Design, October 1961 pp. 3-9.

(6) B. Fleischmann, "Computational Experience With the Algorithm of Balas," Operations Research, vol. 14, no. 1, pp. 153-155, January-February, 1966.

(7) R. J. Freeman, "Computational Experience With a 'Balasian' Integer Programming Algorithm," Operations Research, vol. 14, no. 5, pp. 935-942, September-October, 1966.

(8) A. M. Geoffrion, "Integer Programming by Implicit Enumeration and Balas' Method," SIAM Review, vol. 9, no. 2, pp. 178-190, April, 1967.

(9) A. M. Geoffrion, "An Improved Implicit Enumeration Approach for Integer Programming," The RAND Corporation, Memorandum RM-5644-PR, June, 1968.

(10) F. Glover, "A Multiple Phase-Dual Algorithm for the Zero-One Integer Programming Problem," Operations Research, vol. 13, no. 6, pp. 879-919, November-December, 1965.

(11) R. E. Gomory, "An All-Integer Integer Programming Algorithm," Industrial Scheduling edited by J. R. Muth and G. L. Thompson, Prentice-Hall, 1963.

(12) J. Haldi, "25 Integer Programming Test Problems," Working Paper no. 43, Graduate School of Business, Stanford University, December, 1964.

(13) L. Hellerman, "A Catalog of Three Variable OR-Invert and AND-Invert Logical Circuits," IEEETEC, vol. EC-12, no. 3, pp. 198-223, June, 1963.

(14) E. L. Lawler and D. E. Wood, "Branch-And-Bound Methods: A Survey," Operations Research, vol. 14, no. 4, pp. 699-719, July-August, 1966.

(15) C. E. Lemke and K. Spielberg, "Discrete Search Algorithm for Zero-One and Mixed-Integer Programming," Operations Research, vol. 15, no. 5, pp. 892-914, September-October, 1967.

(16) T. K. Liu, "A Code for Zero-One Integer Linear Programming by Implicit Enumeration," Master Thesis, Department of Computer Science, University of Illinois, 1968.

(17) E. J. McCluskey, Introduction to the Theory of Switching Circuits, New York: McGraw-Hill, 1965.

(18) C. E. Miller, A. W. Tucker and R. A. Zemlin, "Integer Programming Formulation of Traveling Salesman Problems," Journal of ACM, vol. 7, pp. 326-329, 1960.

(19) S. Muroga and T. Ibaraki, "Logical Design of an Optimum Network by Integer Linear Programming-Part I," Report no. 264, Department of Computer Science, University of Illinois, July, 1968.

(20) S. Muroga and T. Ibaraki, "Logical Design of an Optimum Network by Integer Linear Programming-Part II," Report no. 289, Department of Computer Science, University of Illinois, December, 1968.

(21) C. C. Peterson, "Computational Experience With Variants of the Balas Algorithm Applied to the Selection of R and D Projects," Management Science, vol. 13, no. 9, pp. 736-750, May, 1967.